Algoritmos Genéticos Incrementales

Sergio Nesmachnow Pablo Musso Federico Dominioni

Centro de Cálculo, Instituto de Computación,
Facultad de Ingeniería, Universidad de la República, Uruguay

Resumen

Este artículo presenta el estudio, diseño e implementación de un modelo particular de algoritmos genéticos paralelos, los algoritmos genéticos incrementales. La principal ventaja del motor de algoritmos genéticos implementado es su potencialidad de ejecución en entornos no dedicados, detectando la carga generada por los usuarios del entorno y consumiendo de un modo "inteligente" los recursos computacionales disponibles. Se presenta la descripción teórica del modelo de algoritmos genéticos incrementales, los detalles de diseño e implementación del motor desarrollado y su evaluación al aplicarse a la resolución de un complejo problema de diseño de redes de comunicaciones confiables.

Palabras clave: algoritmos genéticos, modelos paralelos, entornos no dedicados.

Destinado al Workshop de Agentes y Sistemas Inteligentes

1. Introducción

Este artículo presenta el estudio, diseño e implementación de un motor de algoritmos genéticos paralelos que sigue la propuesta del modelo de algoritmos genéticos incrementales, presentado por Alba y Troya [3]. El software toma ventaja de la capacidad del modelo, adecuado para ejecutar en entornos heterogéneos, en especial sobre redes de computadores no dedicadas. Basados en el paralelismo de *islas* o de *subpoblaciones con migración* [2], los algoritmos genéticos incrementales mantienen las ventajas de calidad de búsqueda de los modelos centralizados, y permiten la ejecución en paralelo con un número *variable* de islas distribuidas en varios procesadores. La implementación desarrollada adecua el modelo para permitir su ejecución en redes de computadores no dedicadas, adaptándose a condiciones de carga variable del entorno de ejecución compartido, configurando automáticamente el número de islas activas dependiendo de la carga generada por otros usuarios.

Si bien existe una amplia variedad de bibliotecas e implementaciones de modelos de algoritmos genéticos (AG) secuenciales y paralelos, el modelo incremental ha recibido una atención casi nula. Las principales bibliotecas de algoritmos genéticos no ofrecen la posibilidad de manejar un número dinámico de islas y ninguna brinda soporte para configurar el número o la estructura de las islas dependiendo de la carga presente en cada computador. Esta característica es un requisito muy importante para el caso de redes no dedicadas, donde se comparte el uso de un conjunto de recursos computacionales con un número importante de usuarios que realizan variadas tareas en un día habitual de trabajo. Para lograr el objetivo de utilizar racionalmente la mayor cantidad de recursos computacionales sin perturbar el normal trabajo de otros usuarios, la implementación desarrollada detecta la presencia de otros usuarios, evalúa su actividad y requerimientos de poder de cómputo y libera los recursos de ser necesario, eliminando islas del modelo incremental. Asimismo, el monitoreo de actividad identifica la disponibilidad de recursos computacionales libres y asigna a ellos la ejecución de islas previamente detenidas o creadas a demanda del propio modelo.

El resto del documento se organiza del modo que se describe a continuación. La sección 2 presenta el modelo de algoritmos genéticos incrementales, describiendo sus variantes. La sección 3 presenta el análisis del motor, describiendo sus características y sus funcionalidades. Los detalles de diseño e implementación se presentan en la sección 4. La sección 5 ofrece la evaluación del sistema mediante la resolución de un problema de optimización relacionado con el diseño de redes de comunicaciones confiables. Por último, la sección 6 ofrece las conclusiones y propuestas de trabajo actual y futuro.

2. Algoritmos genéticos incrementales

El modelo de algoritmos genéticos incrementales fue propuesto por Alba y Troya en 1995 [3] y el único antecedente conocido de implementación corresponde a un trabajo interno no publicado [4]. Es un modelo híbrido que busca conjuntar las ventajas de los modelos centralizados (buena calidad de búsqueda de soluciones), y las ventajas de los modelos distribuidos (mejora en la diversidad, buenos valores de eficiencia computacional y de tiempo efectivo para hallar buenos resultados).

El modelo se compone de dos tipos de procesos: un monitor central y un conjunto de islas que trabajan cooperativamente en paralelo, conformando un sistema homogéneo ya que todas aplican el mismo procesamiento sobre distintas poblaciones. La única vía de comunicación es a través del proceso monitor central, quien se comunica con las islas y viceversa: no existen migraciones entre islas. Una de las diferencias fundamentales de los algoritmos genéticos incrementales con respecto a los modelos migratorios tradicionales consiste en que el monitor recibe los individuos desde las islas de manera asíncrona. Se genera de este modo una macro-población compuesta por los mejores individuos de cada isla, *de manera incremental* y *no determinista*: la mezcla de individuos provenientes de las islas no guarda un orden determinado respecto a los procesos que los crearon. Las islas no trabajan siempre sobre la misma población, dado que en ciertos puntos de su evolución cada isla recibirá del monitor una nueva población para procesar.

La Figura 1 presenta el esquema de los algoritmos genéticos incrementales.
Según la actividad genética del proceso monitor, y dependiendo de la variación dinámica de la cantidad de islas en el sistema, se distinguen cuatro modelos de algoritmos genéticos incrementales:

- Modelo estático–pasivo: El monitor no realiza trabajo genético. El número de islas se define al iniciar el trabajo y permanece constante hasta el final del algoritmo.
- Modelo estático–activo: El monitor realiza trabajo genético y el número de islas permanece constante hasta la terminación del algoritmo.
- Modelo dinámico–pasivo: El monitor no realiza trabajo genético, y el número de islas puede variar durante la ejecución del algoritmo. Las islas pueden eliminadas, y también es posible la creación de nuevas islas que realicen tareas de búsqueda genética.
- Modelo dinámico–activo: El monitor realiza trabajo genético y el número de islas puede variar durante la ejecución del algoritmo.

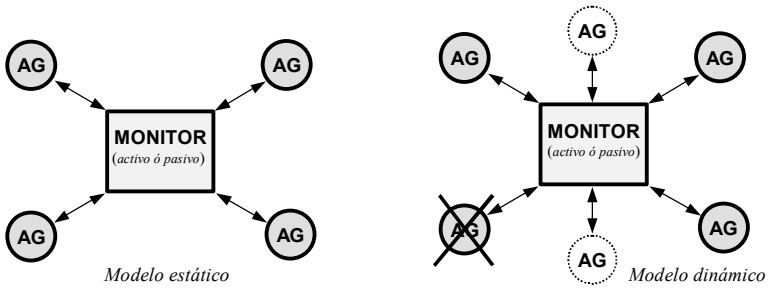

Figura 1. Esquema general de los algoritmos genéticos incrementales

3. El motor de algoritmos genéticos incrementales

Esta sección presenta los requisitos del motor de algoritmos genéticos incrementales, una descripción precisa del modelo implementado y las características generales del sistema desarrollado.

3.1. Requisitos del motor

El principal requisito planteado para el motor de algoritmos genéticos incrementales es la capacidad de ser ejecutado en redes no dedicadas. Por este motivo, es fundamental contar con la creación y la eliminación dinámica de islas, descartándose la utilidad de los modelos estáticos. Los modelos activo y pasivo son semejantes algorítmicamente y la calidad de búsqueda entre un modelo dinámico-activo con N islas y un modelo dinámico-pasivo con N+1 islas será similar para N razonablemente grande. La eficiencia computacional y el tiempo real de ejecución para hallar resultados de calidad razonable deben ser similares para ambos modelos.
Como consecuencia, se decidió fijar como alcance del trabajo el desarrollo de una implementación de algoritmos genéticos incrementales *modelo dinámico-pasivo***.** De todas formas, se ha propuesto también una especificación del modelo dinámico-activo que intenta subsanar ciertas desventajas de la propuesta original, y el motor de algoritmos genéticos incrementales se ha desarrollado teniendo en cuenta su posible extensión para incorporar el modelo activo en el futuro.

3.2. Modelo dinámico-pasivo

3.2.1. Propuesta original

En la propuesta original de modelo dinámico-pasivo [4], el proceso monitor comienza con una población vacía y durante la evolución crea dinámicamente islas con una población de tamaño T_p. En cada generación, las islas calculan su *porcentaje de mejora*, considerando las relaciones entre el mejor valor y el promedio de fitness en dos generaciones sucesivas, y lo comunican al monitor cuando este proceso lo requiere. Por cada individuo generado en una isla se envía una copia al monitor, que los almacena de manera no determinista en su población. Cuando dispone de T_p individuos toma una decisión en función del número actual de islas y de los porcentajes de mejora:

- Si el número actual de islas ejecutando en el sistema es menor que el máximo permitido, entonces el monitor creará una nueva isla con la población incremental.
- Si se encuentra ejecutando la cantidad máxima posible de islas, entonces el monitor le envía la población incremental a la isla de peor porcentaje de mejora.

Cuando una isla recibe una población del monitor, abandona su población y comienza a procesar la nueva población recibida, aplicando el proceso evolutivo del AG. En general, la condición de parada del modelo involucra una cantidad específica de entregas de la población incremental del monitor.

3.2.2. Desventajas de la propuesta original de modelo dinámico-pasivo

La propuesta original de modelo dinámico-pasivo presenta una serie de desventajas:

- La tasa de comunicación entre las islas y el monitor es altísima. Al enviar cada nuevo individuo al monitor se centraliza demasiado la comunicación, degradando la eficiencia computacional del modelo. Además, al comunicarse individuos que no tienen buen fitness, se ocasiona una latencia en la difusión de buenos individuos al resto de las islas.
- Dado que en un modelo de AG generacional se crea toda la población en una generación, se corre el riesgo de no mezclar adecuadamente poblaciones diferentes en el monitor.
- Como cada isla informa su porcentaje de mejora ante solicitudes del proceso monitor, en el caso de utilizar comunicaciones asincrónicas el monitor debe esperar a recibir todos los porcentajes de mejora, o enviar poblaciones basándose en información no reciente.
- Al momento de crear nuevos procesos no se tiene en cuenta que realmente existan recursos computacionales con la capacidad adecuada de ejecutar el nuevo proceso genético.

3.2.3. Modelo dinámico-pasivo propuesto

Considerando las desventajas presentadas del modelo original, se propusieron incorporar ciertas características al modelo dinámico-pasivo.

Se propone reducir el envío de individuos desde las islas al monitor, comunicando T_p/N individuos seleccionados, siendo N la cantidad de islas activas en el sistema. El envío no se realizará en cada generación: un parámetro adicional del modelo determinará la frecuencia de comunicación. Al permitir la evolución independiente de las islas durante cierto número de generaciones, se mejora la eficiencia computacional reduciendo el acceso al cuello de botella de las comunicaciones, y se aumenta la diversidad de la población en el monitor, evitando que predominen los individuos provenientes de una determinada isla. En cada generación las islas enviarán sus porcentajes de mejora al monitor sin necesidad de solicitud previa, permitiendo al proceso monitor consultar de forma inmediata los porcentajes de mejora actualizados de cada isla, sin esperar a recibirlos.

Para permitir su ejecución en ambientes compartidos, se considerará la carga actual de cada host del entorno para determinar si se deberá crear una nueva isla o no, evitando que los hosts sean saturados con procesos genéticos, afectando las tareas de otros potenciales usuarios del entorno de ejecución.

Adicionalmente, se incorporan al modelo flexibilidades para considerar casos específicos que permitan una mejora en el desempeño computacional al ejecutar sobre una red de computadores:

- Cuando el monitor posee T_p individuos y no sea posible crear nuevas islas, enviará los T_p/N "mejores" individuos de su población incremental a la isla con menor porcentaje de mejora, que seleccionará para reemplazar sus "peores" T_p/N individuos.
- Se incorpora la posibilidad de comunicación de material genético entre islas sin pasar por el monitor mediante mecanismos de migración asincrónica.
- En el momento de eliminar una determinada isla, se almacenará la información correspondiente para que el trabajo genético realizado no se pierda. Con este objetivo se definirá un conjunto de *check-points*, especificado por un número de generaciones que se incorpora como parámetro adicional al modelo. En cada *check-point* se almacenará en disco la información (*status*) de las islas, que podrá ser utilizada para reanudar la tarea en el futuro.

- De modo similar, el monitor almacenará su status cada cierto número de envíos de T_p (ó T_p/N) individuos. De esta forma se dispondrá del estado del sistema en conjunto almacenado para ser recuperado en caso de que se interrumpa la ejecución del sistema, continuando con el procesamiento genético desde el último *check-point* disponible.

3.3. Características del sistema

La principal característica del sistema es su capacidad de ejecución sobre *redes de computadoras no dedicadas*, un entorno de tiempo compartido que se utiliza para múltiples fines, donde muchos usuarios trabajan concurrentemente, generando cada uno carga de CPU y memoria sobre el sistema. Para evitar consumir la totalidad de los recursos del entorno, perjudicando el trabajo normal de otros usuarios de la red, la propuesta incorporará la capacidad de detectar automáticamente la carga generada por los usuarios del entorno, permitiendo la creación y eliminación dinámica de procesos genéticos. Específicamente, el motor deberá ser capaz de realizar las siguientes actividades:

- Detectar la presencia de usuarios utilizando un recurso computacional (*host*) y eliminar la cantidad necesaria de procesos genéticos para liberar recursos, salvando su estado de ejecución.
- Detectar la disponibilidad de recursos computacionales. Ante la ausencia de usuarios utilizando un host, se deberá crear la cantidad de procesos genéticos que el sistema considere necesario para aprovechar al máximo los recursos disponibles del entorno de ejecución.
- Poder recuperar el estado de ejecución de todo el sistema, evitando perder el trabajo de búsqueda realizado hasta el momento.

3.4. Formalización del Modelo Dinámico-Pasivo propuesto

Siguiendo la especificación definida por Alba y Troya, la Figura 2 presenta la formalización del modelo y una descripción de sus parámetros.

```
AG_Inc_Din_Pas (AG[], Tp, N∈[Nmin,Nmax], M, Td=alcanzar(P), Monitor=pasivo,
island_send_rate, island_migration_rate, island_migration_size, CARGA_MIN,
CARGA_MAX, check_asynchronous, time_check_process, save_status, proc(Maq)).
```

- `AG[]`: Conjunto de AGs presentes en el modelo (islas).
- `Tp`: Tamaño de la población presente en cada isla del sistema.
- `M`: Cantidad de procesadores que conforman el entorno de ejecución.
- `Td=alcanzar(P)`: Condición de finalización del algoritmo genético incremental.
- `N`: Número de islas activas en el sistema.
- `Nmin`: Número mínimo de islas que se permiten en el sistema en cualquier momento.
- `Nmax`: Número máximo de islas permitidas en el sistema en cualquier momento. Inicialmente `N=Nmáx`, o sea, el sistema comienza a trabajar con la cantidad máxima de procesos genéticos.
- `island_send_rate`: Frecuencia de envío de individuos desde las islas hacia el proceso monitor, medida en generaciones.
- `island_migration_rate`: Frecuencia de migración entre islas, medida en generaciones.
- `island_migration_size`: Tamaño de la migración entre islas.
- `CARGA_MIN`: Carga de un host por encima de la cual no se crearán nuevas islas.
- `CARGA_MAX`: Carga de un host por encima del cual se eliminaran procesos genéticos.
- `check_async`: Frecuencia para chequear recepción de individuos, medida en generaciones
- `time_check_process`: Tiempo (en segundos) que indica la frecuencia de actualización de las lecturas de carga de cada host perteneciente al entorno de ejecución.
- `save_status`: Frecuencia para almacenar el estado de los procesos. Medida en generaciones para las islas y en número de veces que envió T_p (o T_p/N) individuos a las islas para el monitor.
- `proc()`: Lista con la cantidad máxima de procesos genéticos que pueden ejecutar en cada host.

Figura 2: Especificación del modelo dinámico-pasivo propuesto.

3.5. Descripción del sistema

El sistema opera según los seudocódigos que se presentan y describen a continuación.

```
Proceso MONITOR:
Para todo AG hacer iniciar_AG(AGi,i,false,NULL,NULL);  // Iniciar los N procesos genéticos.
nro_envios = 0
Hacer P veces:                    // P = Número de generaciones que realizará cada AG.
   num_ind_Monitor = 0
   Mientras num_ind_Monitor <= Tp
       Recibir(msg)               // Recibir un mensaje.
       msg:nuevoIndividuo(i,Individuo) → // Se recibe individuo: agregar a pob. incremental.
             poblacionMonitor+=individuo
             num_ind_Mon = num_ind_Mon + 1
             Actualizar_Estadísticas()
       msg:porcentaje_mejora(i,porc) → // Se recibe info: actualizar lista de mejoras.
             Lista_mejoras(i)=Porc
       msg:kill_ack(i,path): → agregar(kill_process,path)  // Eliminar isla.
             kill(i);
       Cada time_check_process: → actualizar(Lista_carga).
   fin mientras;
   nro_envios = nro_envios + 1
   Si (nro_envios % save_status==0)       // Guardar status si corresponde.
       guardar_status_disco(path_archivo_status)
   receptor = destino(Lpi,Lpm,crear_nuevo_proc) // Determinar destino de la pob. incremental.
   Si crear_nuevo_proc
       iniciar_ag(receptor,false,Población_Monitor) // Iniciar AG con la pob. incremental.
   Sino
       Pob_env = Seleccionar(Población_Monitor,Tp/N) // Seleccionar individuos para envío.
       nueva_poblacion(AG_receptor,Pobenv)
    fin Si
fin Hacer
Para todo AGi hacer terminar_AG(i);
Reportar resultado y estadísticas.
```

Figura 3: Seudocódigo del proceso monitor.

El monitor almacena en una tabla, actualizada cada *time_check_process* segundos, los datos de carga de cada host, evaluados mediante una combinación lineal entre los porcentajes de uso de procesador y de memoria en el último período, de acuerdo a la expresión presentada por la Ecuación 1.

$$CARGA = peso\_CPU * porcentaje\_carga\_CPU + peso\_memoria * porcentaje\_carga\_memoria$$

Ecuación 1: Modelo lineal para la determinación de la carga de un host.

El consumo de CPU y de memoria son los parámetros más representativos para evaluar la carga en entornos compartidos [5]. El modelo lineal considerado ha proporcionado precisos resultados para determinar la relación entre carga y tiempo de ejecución efectivo de aplicaciones de calculo científico en nuestro entorno de trabajo [9]. Los pesos fueron determinados mediante la evaluación empírica del tiempo de ejecución de un programa determinista bajo diferentes condiciones de carga generadas por programas estándares del entorno de trabajo compartido (compilaciones, aplicaciones de oficina, scripts utilitarios, etc.), ajustando mediante aproximación de mínimos cuadrados los valores correspondientes a las contribuciones de carga de CPU y memoria respectivamente.

De acuerdo a los valores de tolerancia definidos, el proceso monitor eliminará islas en aquellos hosts cuyo valor de carga supera *CARGA_MAX*. Para evitar eliminar procesos cuando la carga de usuario decrece, se realiza una extrapolación lineal a partir de los valores de carga anterior y actual: la eliminación se hará cuando la predicción de carga futura supere al umbral *CARGA_MAX*. La predicción de carga mediante una extrapolación lineal ha mostrado una capacidad aceptable sobre nuestro entorno de ejecución, mejorando los valores de predicción ingenua (*naive*) usados por los algoritmos de despacho de las bibliotecas de programación paralela, tal como se ha reportado en [9].

Si existen varias islas ejecutando en un host cuyos valores de carga superan el umbral, se eliminarán de forma ordenada cada *time_check_process* segundos, considerando sus porcentajes de mejora.

El monitor almacena en una lista el camino absoluto del archivo con el status de los procesos genéticos eliminados. La creación por parte del monitor de un nuevo proceso genético en un host *H* depende de tres condiciones: que el número de islas sea menor que el valor indicado por el parámetro `Nmax`, que el número de procesos genéticos en el host *H* sea menor que el parámetro `proc(H)`, y que el valor de carga del host *H* sea menor que el parámetro `CARGA_MIN`. Para evitar crear un proceso que deberá ser eliminado inmediatamente, se predice la carga del host *H* en los siguientes `time_check_process` segundos, extrapolando linealmente a partir de los últimos dos valores de carga. La Figura 4 presenta las regiones delimitadas por los umbrales de carga y las acciones de creación, eliminación de procesos y ejecución "en régimen" (sin creación ni eliminación de islas).

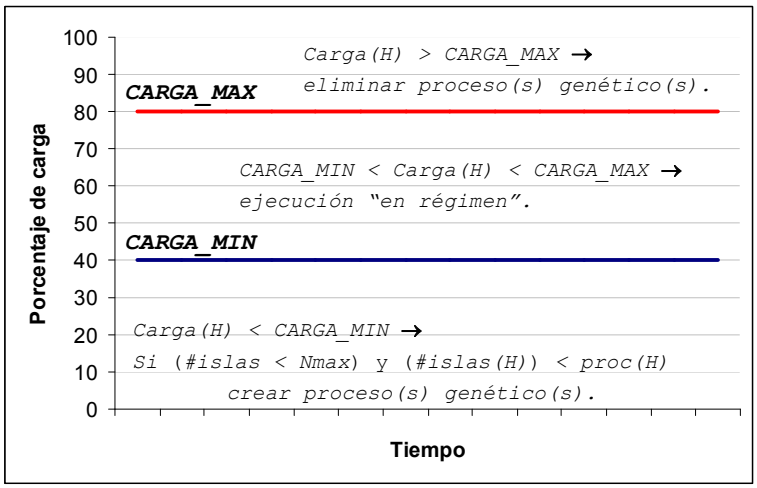

Figura 4: Regiones delimitadas por umbrales de carga y acciones correspondientes.

Cada `save_status` envíos hacia las islas, el monitor almacena su estado a disco, definiendo un punto de sincronización para reanudar la ejecución en un momento posterior.

```
AGi(Agi, i, leerarchivo, path, población)
Si(leer_de_archivo)
   Población_Isla = leer_población_de_archivo(path)   // Retomar ejecución previa.
   Si (no_leer_de_archivo) y (población == vacío)
   Población_Isla = Generar_Población_AG()            // Iniciar ejecución.
Sino
   Población_Isla = población;                        // Sustituir por pob. incremental.
Hasta recibir el mensaje terminar_ag(i) hacer:
   Mientras no se reciba mensaje(msg) hacer:
       Evolucionar(Población_Isla)                    // Aplicar mecanismo evolutivo.
       Si (generacion % migration_rate == 0)          // Migración
              dest = destinoMigracion();
              Pob_Mig = Seleccionar(Población_Isla, migration_size)
              Enviar Migracion(Pob_Mig ,dest)
       Si (generacion % check_async == 0)      // Chequear por inmigrantes
              recibir_Migracion(origen, ind_ mig, i)
              reemplazar(Población_Isla , ind_mig)
              Actualizar_porcentaje_de_mejora()
       Enviar porcentajemejora(i,porcentaje_de_mejora).
       Si (generacion % save_status == 0)       // Salvar status a disco si corresponde.
              guardar_status_disco(path_archivo_status)
       Si (generacion % island_send_rate == 0) // Enviar individuos al monitor si corresponde.
              Inds = Seleccionar(Población_Isla, island_send_size)
              Enviar_individuos(i, Inds)
   Fin Mientras
   msg:nueva_población(nueva_población)           // Se recibe nueva población.
       remplazar(Población_Isla, nueva_población) // Reemplazar los peores individuos
   msg:kill_island
       guardar_status_disco(path_archivo_status); // Salvar status y envíar kill_ack
       envíar_kill_ack(i,path_archivo_status)
```

Figura 5: Seudocódigo de los procesos isla.

Las islas realizan la búsqueda poblacional de un AG tradicional, en base a operadores evolutivos. La población inicial puede leerse desde un archivo (si se reanuda una ejecución previamente truncada), puede recibirse desde el proceso monitor (si se crea un nuevo proceso isla con una población incremental), o puede generarse aleatoriamente mediante el procedimiento de inicialización.

Se incorpora la posibilidad de intercambio de material genético entre islas a través de migraciones asincrónicas: cada `migration_rate` generaciones, una isla selecciona `island_migration_size` individuos y los envía a su vecino más cercano, considerando a los procesos genéticos conectados de acuerdo a una topología de migración que corresponde a un anillo unidireccional. Cada isla verifica cada `check_async` generaciones la recepción de individuos inmigrantes y efectuará el reemplazo manteniendo el criterio de mantener los mejores individuos en la población.
Cuando recibe un mensaje de `nueva_poblacion`, la isla procede a reemplazar la totalidad de la población actual por la población incremental que le envía el proceso monitor.
El procedimiento iterativo se realiza hasta recibir un mensaje de finalización (`kill_island`). Luego se almacena el estado para evitar la perdida del trabajo realizado, estableciendo un check-point para reanudar la evolución en caso de disponer de recursos computacionales en el futuro. A continuación, la isla envía la confirmación de finalización (`kill_ack`), y finaliza su ejecución.

4. Diseño e Implementación

El sistema desarrollado se compone de dos grandes módulos. Un módulo brinda las funcionalidades del modelo de algoritmos genéticos incrementales descrito en la sección anterior, implementando el proceso maestro y los procesos islas. Por otra parte, un segundo módulo se encarga de monitorear la carga de los hosts que componen el entorno de ejecución, proporcionando la información necesaria al proceso monitor para decidir si corresponde crear o eliminar procesos en el entorno.

4.1. El módulo de algoritmos genéticos

El modelo desarrollado se basa en un monitor que no realiza trabajo genético. Su tarea consiste en tomar decisiones sobre la creación o eliminación de procesos genéticos según la carga generada por los usuarios en cada host y al porcentaje de mejora de cada proceso genético existente en el sistema.
El módulo de algoritmos genéticos incrementales se diseñó extendiendo MALLBA, una biblioteca que proporciona esqueletos de software que implementan algoritmos de optimización [1]. Los detalles de los algoritmos, la interacción con el problema y el paralelismo se implementan mediante clases C++ que abstraen a las entidades involucradas en cada método de resolución:

- Las clases provistas implementan aspectos cada algoritmo, independizándose del problema. Las principales clases provistas son `Solver` (el algoritmo) y `SetUpParams` (fija parámetros).
- Las clases requeridas especifican la información relevante del problema a resolver. Cada esqueleto incluye las clases requeridas `Problem` y `Solution` que encapsulan las entidades dependientes del problema necesarias para los métodos de resolución.

La biblioteca MALLBA está disponible públicamente en **`http://neo.lcc.uma.es/mallba`**.
La comunicación entre los procesos isla y el proceso monitor se realiza utilizando la biblioteca de desarrollo de programación paralela y distribuida MPI [6].

4.2. El módulo medidor de carga

El módulo medidor de carga o *medidor de performance* realiza lecturas de uso de CPU y memoria para cada host del entorno de ejecución. Se implementó como una aplicación cliente/servidor con comunicación entre procesos mediante invocación de procedimientos remotos (RPC). La obtención de datos de carga es una operación en que el monitor invoca funciones remotas o servicios sobre cada host del entorno de ejecución. Se seleccionó RPC como mecanismo de comunicación por su simplicidad conceptual, pero el diseño prevé utilizar cualquier otro mecanismo de comunicación, ya que abstrae y encapsula toda la lógica relacionada con la tecnología de comunicación.

Un proceso *servidor de carga* ejecuta en cada host del entorno de ejecución, evaluando su carga. El proceso *cliente de carga*, presente en el host que ejecuta al monitor, se comunica remotamente con cada servidor de carga para obtener los valores de cada host del entorno de ejecución.
Los módulos del sistema se relacionan de acuerdo al esquema que se presenta en la Figura 6. El monitor dispone de una colección de clientes de carga, uno por cada host del entorno de ejecución. Existe una independencia total entre cada proceso isla ejecutando en un host y el servidor de carga correspondiente, que sólo conoce el número de islas en ejecución en el host que monitorea.
Resumiendo, si bien el diseño de los módulos encapsula y abstrae el mecanismo de comunicación utilizado para no ligarse a una tecnología en particular, el sistema utiliza dos tipos de comunicación:

- una comunicación entre procesos genéticos (monitor e islas) que se realiza mediante MPI.
- una comunicación entre el monitor (a través de su colección de clientes de carga) y los servidores de carga, utilizando RPC.

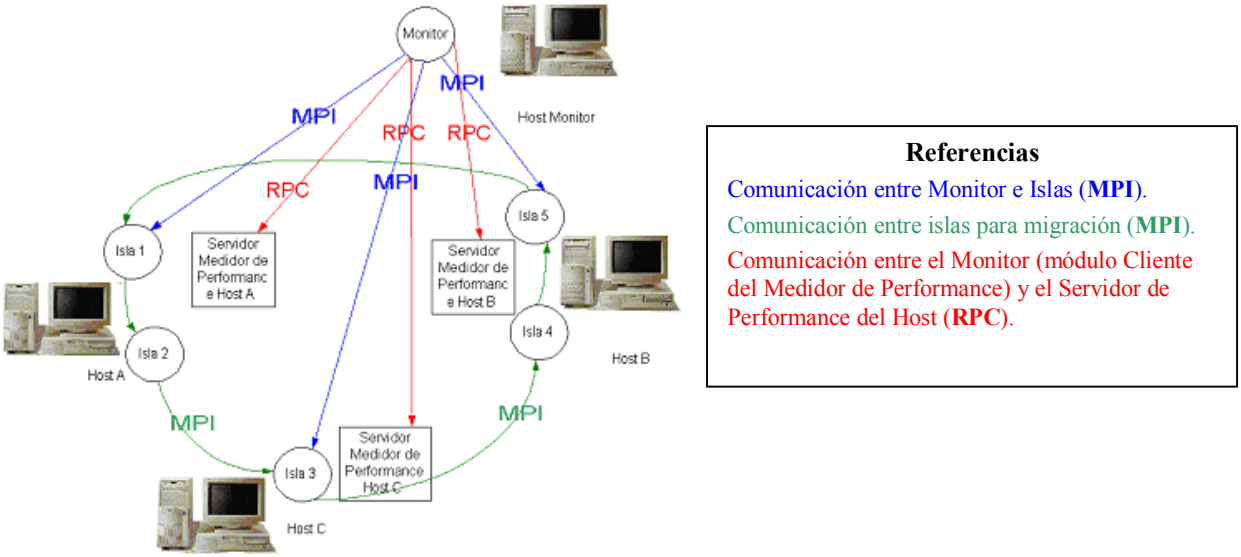

Figura 6: Arquitectura del Sistema

5. Evaluación del sistema

Para verificar la correctitud de la implementación y evaluar su capacidad de trabajo sobre un entorno no dedicado respetando las consideraciones de diseño, se utilizó el motor de algoritmos genéticos incrementales para la resolución del problema de Steiner generalizado (GSP).
El GSP propone optimizar el diseño de una red de comunicaciones respetando requisitos de conectividad entre nodos distinguidos denominados terminales. El problema ha sido abordado previamente por nuestro grupo de trabajo utilizando el modelo de islas de MALLBA sobre un entorno de red de computadoras dedicado [7,8]. La evaluación experimental se orientó a comparar la calidad de resultados y la eficiencia computacional del modelo de algoritmos genéticos incrementales con los resultados previamente obtenidos. Se trabajó sobre las tres instancias del problema abordadas en trabajos previos (*grafo_50_15*, *grafo_75_25* y *grafo_100_10*, nombres que refieren al número de nodos y de terminales), que proporcionan diferentes niveles de complejidad para problemas representativos del diseño de redes de comunicaciones de mediano tamaño.

5.1. Plataforma de ejecución

Los experimentos de evaluación se realizaron sobre un clúster no dedicado compuesto por 8 computadores heterogéneos, con procesadores Pentium II y Pentium III de entre 400 MHz y 1.2 GHz y entre 256 y 1024 MB de RAM, con sistema operativo Linux Suse 8.2 y conectadas por Ethernet común de 10 Mbits.

5.2. Evaluación de resultados

La Tabla 1 resume los resultados obtenidos por el modelo de algoritmos genéticos incrementales, presentando las mejores soluciones, valores promedio y desviación estándar sobre 10 ejecuciones para cada instancia de prueba considerada. Se comparan los valores con los resultados del modelo de islas ejecutando sobre un cluster dedicado de 8 computadores.

	Mejor AGI	***Promedio AGI***	***Desv. Est.***	***Mejor Islas***	***Promedio Islas***
grafo 50_15	9498.65	9410.17	65.10	9535.29	9393.25
grafo 75_25	5444.69	5344.75	44.86	5447.23	5379.41
grafo 100_10	4536	4483.25	37.17	4537	4489

Tabla 1: Comparación de resultados.

La Tabla 1 muestra que el modelo de algoritmos genéticos incrementales alcanza resultados similares al modelo tradicional de islas implementado por MALLBA para las instancias estudiadas.

5.3. Evaluación de eficiencia computacional

Para evaluar la eficiencia computacional del modelo implementado, se calculó el *speedup* y la *eficiencia* del algoritmo paralelo. El *speedup* (S_M), relaciona el tiempo medio de ejecución al usar un solo procesador (T_1) con el tiempo medio al utilizar M procesadores (T_M) según la Ecuación 2.1. La *eficiencia* (E_M), normaliza el valor del speedup respecto a los recursos computacionales utilizados, según la Ecuación 2.2.

$$S_M = \frac{T_1}{T_M} \quad (2.1) \qquad E_M = \frac{S_M}{M} \quad (2.2)$$

Ecuación 2: Definición de speedup y eficiencia.

	grafo 50_15		***grafo 75_25***		***grafo 100_10***	
	S_8	E_8	S_8	E_8	S_8	E_8
AG Incrementales	2.64	0.33	4.65	0.58	2.48	0.31
AG Islas [8]	5.04	0.63	5.20	0.65	5.2	0.65

Tabla 2: Resultados comparativos de eficiencia computacional

La Tabla 2 presenta los resultados comparativos de speedup y eficiencia de los modelos estudiados, utilizando 8 procesadores para resolver cada instancia del problema. Se observa que el modelo de islas supera notoriamente al modelo incremental en las instancias menos complejas, y que el modelo incremental es competitivo en la instancia de mayor complejidad (el grafo con mayor cantidad de terminales). La degradación de la eficiencia del modelo incremental en las instancias menos complejas se debe a dos factores: la influencia del trabajo en el entorno no dedicado, que agrega algoritmia para manejar la interacción con otros usuarios; y el incremento de las comunicaciones debido a la transferencia de individuos entre islas y monitor para formar la población incremental. La influencia de estos factores disminuye al aumentar la complejidad de los problemas; por este motivo el modelo incremental logra alcanzar valores aceptables de eficiencia para el *grafo_75_25*.

Se estudió la escalabilidad del modelo incremental para resolver la instancia *grafo_75_25*, limitada a los recursos computacionales disponibles (8 procesadores). Los resultados se ofrecen en la Tabla 3 y en la Figura 7, mostrando el comportamiento de speedup sublineal del modelo incremental. Los valores de eficiencia disminuyen al usar 6 y 8 procesadores, degradándose la performance a medida que se agregan recursos computacionales al incrementarse las comunicaciones entre islas y monitor.

procesadores	***1***		***2***		***4***		***6***		***8***	
	S_1	E_1	S_2	E_2	S_4	E_4	S_6	E_6	S_8	E_8
AG Incrementales	1	1	2.03	1.02	3.64	0.58	4.42	0.31	4.65	0.31

Tabla 3: Resultados de escalabilidad.

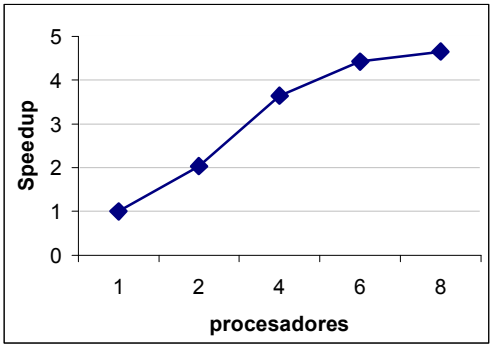

Figura 7: Evaluación gráfica de la escalabilidad.

5.4. Interacción con otros usuarios

Las pruebas de evaluación de eficiencia computacional se realizaron en un entorno no dedicado, interactuando con procesos de otros usuarios. La lógica del sistema detecta la presencia de usuarios y toma acciones para no interferir con el uso habitual de los recursos computacionales. Para evaluar la correctitud de esta funcionalidad, se monitoreó el tiempo promedio que cada ejecución del modelo incremental trabajó por encima del umbral de carga especificado en la configuración. Sobre las 30 ejecuciones realizadas, el modelo compartió CPU con otros usuarios en un tiempo promedio inferior al 5% del tiempo total de ejecución, un valor muy razonable, que permite concluir que el mecanismo que detecta usuarios y libera los recursos computacionales funciona correctamente.

5.5. Conclusiones de la evaluación del sistema

La evaluación permite concluir que el sistema cumple sus especificaciones y es capaz de obtener resultados similares a los del modelo distribuido estándar de algoritmos genéticos. La eficiencia computacional se degrada como consecuencia de la comunicación excesiva entre islas y monitor, pero su efecto se hace menos influyente al aumentar la complejidad del problema a resolver.
La principal característica del modelo de algoritmos genéticos incrementales es su *capacidad de ejecutar la sobre un cluster no dedicado, sin perturbar el trabajo de otros usuarios*. Esta cualidad lo posiciona como una herramienta muy importante para el desarrollo e investigación en nuestro entorno de trabajo, donde no se dispone de recursos computacionales dedicados y es necesario utilizar alternativas –como la red local de trabajo y enseñanza– para resolver problemas complejos.

6. Conclusiones y trabajo futuro

Este trabajo presenta el diseño, la implementación y las pruebas de verificación y evaluación de un modelo de algoritmos genéticos del cual no se tienen antecedentes recientes de implementación. Se incorporó un mecanismo de detección de usuarios en la red que permite la creación y eliminación de procesos genéticos, posibilitando la ejecución en un entorno no dedicado. El modelo se mostró capaz de alcanzar una buena calidad de resultados, y aceptables valores de eficiencia computacional y escalabilidad al resolver un problema de optimización que modela el diseño de redes de comunicaciones confiables. La detección de usuarios funcionó adecuadamente, permitiendo la ejecución del modelo sin perturbar la normal operativa de la red de recursos computacionales.

Algunos aspectos del trabajo han dejado líneas abiertas para abordar en el futuro inmediato. La creación y eliminación de procesos puede ser mejorada cuando se disponga de una implementación de MPI que incluya la primitiva para lanzar procesos *spawn*. Esta funcionalidad está especificada en el estándar MPI2 pero las implementaciones actuales de la biblioteca no la incluyen, restringiendo el manejo de creación de procesos. El diseño del sistema contempla la implementación del modelo incremental activo, en la que se trabaja actualmente. Por último, se prevé utilizar el sistema diseñado para la resolución de otros problemas de optimización NP difíciles, con el objetivo de complementar la evaluación de la calidad de resultados y la eficiencia computacional del modelo.

Referencias bibliográficas

[1] Alba E., Cotta, C. Optimización en entornos geográficamente distribuidos. Proyecto MALLBA. Actas del Primer Congreso Español de Algoritmos Evolutivos y Bioinspirados, pp. 38–45, 2002.

[2] Alba E., Tomassini M. *Parallelism and Genetic Algorithms*. IEEE Transactions on Evolutionary Computation **6** (5), pp. 443–462, 2002.

[3] E. Alba, J.M. Troya, Algoritmos Genéticos Incrementales, Informe Técnico ITI 95-15, Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, 1995.

[4] Dorado, Antonio. Técnicas heurísticas distribuidas frente a técnicas tradicionales para la resolución de problemas NP completos. Memoria de proyecto de fin de carrera, Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, julio de 1997.

[5] Kunz, T. The influence of different workload descriptions on a heuristic load balancing scheme, *IEEE transactions on software engineering*, pp. 725–730, 1991.

[6] MPI Forum Home Page. Disponible en línea **http://www.mpi-forum.org**. Consultada julio 2005.

[7] Nesmachnow, S. Algoritmos Genéticos Paralelos y su Aplicación al Diseño de Redes de Comunicaciones Confiables. Tesis de Maestría en Informática, PEDECIBA Informática, Uruguay, 2004.

[8] Nesmachnow, S., Cancela, H., Alba, E. Técnicas Evolutivas Aplicadas al Diseño de Redes de Comunicaciones Confiables. Actas del Tercer Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB '04), pp. 388–395.

[9] Nesmachnow, S., López, C., López, A. Towards Proactive Network Load Management for Distributed Parallel Programming. IEEE Latin American Network Operations and Management Symposium (LANOMS '99). Río de Janeiro, Brasil, 1999.